Audio Programming 2

Johan Pauwels

johan.pauwels@uwl.ac.uk

On Today’s Programme

Using libraries

Reading audio files

Code libraries

- What?
  - Third-party collections of code providing reusable functionality
- Types
  - Dynamic/shared libraries
  - Static libraries
  - Source code libraries

Library types

- Common to static and shared libraries: consist of
  - Header file containing public interface (API)
  - Binary blob containing compiled implementation
- Static library (*.a/*.lib)
  - Relevant part of binary gets linked into your program/library
  - + Library not needed after compilation: no problem of missing libraries at runtime
  - - Possible duplication of machine code, leading to larger file sizes
- Shared/dynamic library (*.dylib/*.so/*.dll)
  - Binary gets read at runtime, so needs to be present
  - + Large libraries can be shared by multiple programs
  - + Updates possible without recompiling programs that use it (ABI compatibility)
  - - Needs to be present, potential version and naming conflicts (“dll hell”), space inefficient if only one program needs small part of library

Library types

- Common to static and shared libraries:
  - Need matching OS and architecture
  - Can be used to “hide” implementation to protect commercial code
  - Saves compiling never-changing code
- Source code libraries
  - Simply full source given (no hiding possible)
  - Sometimes necessary for technical reasons (template libraries)
  - + Most flexible, compile on any OS and architecture (assuming portable code), modify to your needs
  - - Longer compilation time, wasteful if library code remains unchanged
- Licencing
  - Always check
  - LGPL allows dynamic linking without reciprocal requirement, but not static linking (simplified explanation, IANAL)

Working with libraries

- Required configuration
  - The interface with declarations of classes, variables, functions, constants, structures, definitions needs to be known by the **compiler**
    - Include the header in your code: #include “library.h”
    - Give folder where headers can be found (search path): -I/usr/local/include
(avoids system dependent info in code)
    - Potentially multiple headers in same folder
    - Hierarchy of headers possible: #include “base/header.h”, then specify parent folder of “base”
  - The machine code with library implementation is only required by the **linker** (distinction rarely relevant in practice)
    - Path often split into filename and folder (search path)
-llibrary –L/usr/local/lib
(again for portability between systems and multiple libraries in one folder)
    - Not absolutely necessary, specifying absolute path also possible

CLion and CMake

- CMake: cross-platform make, IDE-agnostic build system **generator**
  - Can create project files for range of different IDEs

```
mkdir xcode-build
cd xcode-build
cmake .. -G Xcode
```

- Configuration by writing text files “CMakeLists.txt”
- CLion: no native project file format, instead close CMake integration
  - No GUI for configuring library options
  - Instead edit CMakeLists.txt directly

CMake essentials

- Autogenerated boilerplate based on creation dialog

```
cmake_minimum_required(VERSION 3.17)
project(AudioFile)
set(CMAKE_CXX_STANDARD 14)
add_executable(AudioFile main.cpp AudioFile.cpp AudioFile.h)
# create executable “AudioFile” by compiling the following sources
```

- Add library options

```
include_directories("/usr/local/include") # includes search path
target_link_libraries(AudioFile "/usr/local/lib/libsndfile.dylib")
# link library at given path to ”AudioFile” executable target
```

Further CMake

- Essential tutorial: https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html
- Further Cmake capabilities (testing, packaging, …) https://cmake.org/cmake/help/latest/guide/tutorial/index.html
- Functionality to search for library include and link paths (instead of hardcoding, to increase portability) https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html#link-libs

libsndfile

- A library designed to allow the reading and writing of many different sampled sound file formats
- Makes abstraction of actual file format on disk
  - reading into float gives samples as floating point [-1, 1], regardless of format on disk (WAVE files often encodes as signed 16-bit integers)
  - Produces interleaved channels
- Docs:
  - http://libsndfile.github.io/libsndfile/api.html

Exercise – AudioFile class

- Write a C++ class that uses libsndfile and provides the following public methods:

```
class AudioFile {
   AudioFile(const std::string& file_path,
                     const bool interleaved=true);
  ~AudioFile();
  const sf_count_t getNumFrames();
  const int getNumChannels();
  const int getSampleRate();
  const sf_count_t readAllFrames(float* buffer);
};
```